

Fig. 1.13 Lateral view of the macaque brain showing the connections from the primary somatosensory cortex, areas 1, 2 and 3, via area 5 in the parietal cortex, to area 7b. Abbreviations as in Fig. 1.9.

area 5, and thus to parietal cortex area 7b (see Figs 1.8 and 1.13, and Section 10.5). Along this pathway more complex representations are formed of objects touched (Iwamura, 1993), and of where the limbs are in relation to the body (by combining information from different joint proprioceptors). In area 7, some neurons respond to visual and to related somatosensory stimuli. Outputs from the parietal cortex project to the premotor areas and to the basal ganglia, which both provide routes to behavioural output. In addition, the parietal cortex projects to the dorsolateral prefrontal cortex, which provides a short term or working memory for limb responses, as shown by the effects of lesions to the dorsolateral prefrontal cortex, and by recordings of neuronal activity in it during delayed response tasks (Goldman-Rakic, 1996). The dorsolateral prefrontal cortex can influence behaviour through basal ganglia outputs, and through outputs to premotor areas.

The hippocampus receives inputs from both the 'what' and the 'where' systems (see Chapter 6 and Fig. 1.12). By rapidly learning associations between conjunctive inputs in these systems, it is able to form memories of particular events occurring in particular places at particular times. To do this, it needs to store whatever is being represented in each of many cortical areas at a given time, and to later recall the whole memory from a part of it. The types of network it contains which are involved in this simple memory function are described in Chapter 6.

With this overview of some of the main processing streams in the cerebral cortex, it is now time to consider in Chapters 2–5 the operation of some fundamental types of biologically plausible network. Then in Chapter 6–10 we will consider how these networks may contribute to the particular functions being performed by different brain regions.

## 2 Pattern association memory

A fundamental operation of most nervous systems is to learn to associate a first stimulus with a second which occurs at about the same time, and to retrieve the second stimulus when the first is presented. The first stimulus might be the sight of food, and the second stimulus the taste of food. After the association has been learned, the sight of food would enable its taste to be retrieved. In classical conditioning, the taste of food might elicit an unconditioned response of salivation, and if the sight of the food is paired with its taste, then the sight of that food would by learning come to produce salivation. More abstractly, if one idea is associated by learning with a second, then when the first idea occurs again, the second idea will tend to be associatively retrieved.

### 2.1 Architecture and operation

The essential elements necessary for pattern association, forming what could be called a prototypical pattern associator network, are shown in Fig. 2.1. What we have called the second or unconditioned stimulus pattern is applied through unmodifiable synapses generating an input to each unit which, being external with respect to the synaptic matrix we focus on, we can call the external input  $e_i$  for the  $i$ th neuron. (We can also treat this as a vector,  $e$ , as indicated in the legend to Fig. 2.1. Vectors and simple operations performed with them are summarized in Appendix A1). This unconditioned stimulus is dominant in producing or forcing the firing of the output neurons ( $r_i$  for the  $i$ th neuron, or the vector  $r$ ). At the same time, the first or conditioned stimulus pattern  $r'_j$  for the  $j$ th axon (or equivalently the vector  $r'$ ) present on the horizontally running axons in Fig. 2.1 is applied through *modifiable* synapses  $w_{ij}$  to the dendrites of the output neurons. The synapses are modifiable in such a way that if there is presynaptic firing on an input axon  $r'_j$  paired during learning with postsynaptic activity on neuron  $i$ , then the strength or weight  $w_{ij}$  between that axon and the dendrite increases. This simple learning rule is often called the Hebb rule, after Donald Hebb who in 1949 formulated the hypothesis that if the firing of one neuron was regularly associated with another, then the strength of the synapse or synapses between the neurons should increase in strength. (In fact, the terms in which Hebb put the hypothesis were a little different from an association memory, in that he stated that if one neuron regularly comes to elicit firing in another, then the strength of the synapses should increase. He had in mind the building of what he called cell assemblies. In a pattern associator, the conditioned stimulus need not produce before learning any significant activation of the output neurons.

The connections must simply increase if there is associated pre- and postsynaptic firing when, in pattern association, most of the postsynaptic firing is being produced by a different input.) After learning, presenting the pattern  $r'$  on the input axons will activate the dendrite through the strengthened synapses. If the cue or conditioned stimulus pattern is the same as that learned, the postsynaptic neurons will be activated, even in the absence of the external or unconditioned input, as each of the firing axons produces through a strengthened synapse some activation of the postsynaptic element, the dendrite. The total activation  $h_i$  of each postsynaptic neuron  $i$  is then the sum of such individual activations. In this way, the 'correct' output neurons, that is those activated during learning, can end up being the ones most strongly activated, and the second or unconditioned stimulus can be effectively recalled. The recall is best when only strong activation of the postsynaptic neuron produces firing, that is if there is a threshold for firing, just like real neurons. The reasons for this arise when many associations are stored in the memory, as will soon be shown.

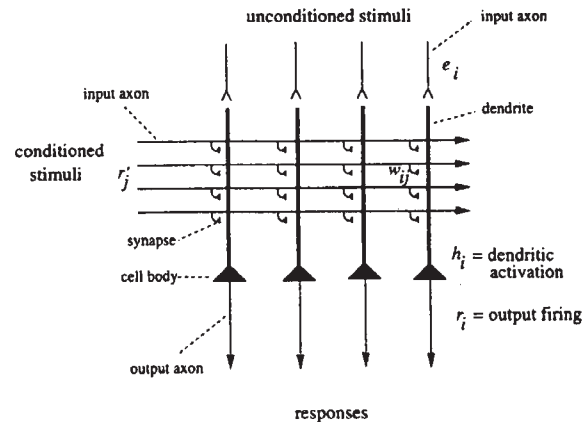


Fig. 2.1 A pattern association memory. An unconditioned stimulus has activity or firing rate  $e_i$  for the  $i$ th neuron, and produces firing  $r_i$  of the  $i$ th neuron. An unconditioned stimulus may be treated as a vector, across the set of neurons indexed by  $i$ , of activity  $e$ . The firing rate response can also be thought of as a vector of firing  $r$ . The conditioned stimuli have activity or firing rate  $r'_j$  for the  $j$ th axon, which can also be treated as a vector  $r'$ .

Next we introduce a more precise description of the above by writing down explicit mathematical rules for the operation of the simple network model of Fig. 2.1, which will help us to understand how pattern association memories in general operate. (In this description we introduce simple vector operations, and, for those who are not familiar with these, have provided in Appendix A1 a concise summary for later reference.) We have denoted above a conditioned stimulus input pattern as  $r'$ . Each of the axons has a firing rate, and if we count or index through the axons using the subscript  $j$ , the firing rate of the first axon is  $r'_1$ , of the second  $r'_2$ , of the  $j$ th  $r'_j$ , etc. The whole set of axons forms a vector, which is just an ordered (1, 2, 3, etc.) set of elements. The firing rate of each axon  $r'_j$  is one element of the firing rate vector  $r'$ . Similarly, using  $i$  as the index, we can denote the firing rate of any output neuron as  $r_i$ , and the firing rate output vector as  $r$ . With this terminology, we can then identify any synapse

onto neuron  $i$  from neuron  $j$  as  $w_{ij}$  (see Fig. 2.1). In this book, the first index,  $i$ , always refers to the receiving neuron (and thus signifies a dendrite), while the second index,  $j$ , refers to the sending neuron (and thus signifies a conditioned stimulus input axon in Fig. 2.1). We can now specify the learning and retrieval operations as follows:

### 2.1.1 Learning

The firing rate of every output neuron is forced to a value determined by the unconditioned (or external or forcing stimulus) input. In our simple model this means that for any one neuron  $i$ ,

$$r_i = f(e_i) \quad (2.1)$$

which indicates that the firing rate is a function of the dendritic activation, taken in this case to reduce essentially to that resulting from the external forcing input (see Fig. 2.1). The function  $f$  is called the activation function (see Fig. 1.3), and its precise form is irrelevant, at least during this learning phase. For example, the function at its simplest could be taken to be linear, so that the firing rate would be just proportional to the activation.

The Hebb rule can then be written as follows:

$$\delta w_{ij} = k r_i r'_j \quad (2.2)$$

where  $\delta w_{ij}$  is the change of the synaptic weight  $w_{ij}$  which results from the simultaneous (or conjunctive) presence of presynaptic firing  $r'_j$  and postsynaptic firing or activation  $r_i$ , and  $k$  is a learning rate constant which specifies how much the synapses alter on any one pairing.

The Hebb rule is expressed in this multiplicative form to reflect the idea that *both* presynaptic and postsynaptic activity must be present for the synapses to increase in strength. The multiplicative form also reflects the idea that strong pre- and postsynaptic firing will produce a larger change of synaptic weight than smaller firing rates. It is also assumed for now that before any learning takes place, the synaptic strengths are small in relation to the changes that can be produced during Hebbian learning. We will see that this assumption can be relaxed later when a modified Hebb rule is introduced that can lead to a reduction in synaptic strength under some conditions.

### 2.1.2 Recall

When the conditioned stimulus is present on the input axons, the total activation  $h_i$  of a neuron  $i$  is the sum of all the activations produced through each strengthened synapse  $w_{ij}$  by each active neuron  $r'_j$ . We can express this as

$$h_i = \sum_j r'_j w_{ij} \quad (2.3)$$

where  $\sum_j$  indicates that the sum is over the  $C$  input axons (or connections) indexed by  $j$ . The multiplicative form here indicates that activation should be produced by an axon only

if it is firing, and only if it is connected to the dendrite by a strengthened synapse. It also indicates that the strength of the activation reflects how fast the axon  $r_j$  is firing, and how strong the synapse  $w_{ij}$  is. The sum of all such activations expresses the idea that summation (of synaptic currents in real neurons) occurs along the length of the dendrite, to produce activation at the cell body, where the activation  $h_i$  is converted into firing  $r_i$ . This conversion can be expressed as

$$r_i = f(h_i) \tag{2.4}$$

where the function  $f$  is again the activation function. The form of the function now becomes more important. Real neurons have thresholds, with firing occurring only if the activation is above the threshold. A threshold linear activation function is shown in Fig. 1.3b. This has been useful in formal analysis of the properties of neural networks. Neurons also have firing rates which become saturated at a maximum rate, and we could express this as the sigmoid activation function shown in Fig. 1.3c. Yet another simple activation function, used in some models of neural networks, is the binary threshold function (Fig. 1.3d), which indicates that if the activation is below threshold, there is no firing, and that if the activation is above threshold, the neuron fires maximally. Whatever the exact shape of the activation function, some non-linearity is an advantage, for it enables small activations produced by interfering memories to be minimized, and it can enable neurons to perform logical operations, such as to fire or respond only if two or more sets of inputs are present simultaneously.

### 2.2 A simple model

An example of these learning and recall operations is provided in a very simple form as follows. The neurons will have simple firing rates, which can be 0 to represent no activity, and 1 to indicate high firing. They are thus binary neurons, which can assume one of two firing rates. If we have a pattern associator with six input axons and four output neurons, we could represent the network before learning, with the same layout as Fig. 2.1, as

		UCS			
CS		1	1	0	0
		↓	↓	↓	↓
1→		0	0	0	0
0→		0	0	0	0
1→		0	0	0	0
0→		0	0	0	0
1→		0	0	0	0
0→		0	0	0	0

Fig. 2.2

where  $r'$  or the conditioned stimulus (CS) is 101010, and  $r$  or the firing produced by the unconditioned stimulus (UCS) is 1100. (The arrows indicate the flow of signals.) The synaptic weights are initially all 0. After pairing the CS with the UCS during one learning trial, some of the synaptic weights will be incremented according to Eq. 2.2, so that after learning this pair the synaptic weights will become

		UCS			
CS		1	1	0	0
		↓	↓	↓	↓
1→		1	1	0	0
0→		0	0	0	0
1→		1	1	0	0
0→		0	0	0	0
1→		1	1	0	0
0→		0	0	0	0

Fig. 2.3

We can represent what happens during recall, when for example we present the CS that has been learned, as follows:

		CS				
1→		1	1	0	0	
0→		0	0	0	0	
1→		1	1	0	0	
0→		0	0	0	0	
1→		1	1	0	0	
0→		0	0	0	0	
		↓	↓	↓	↓	
		3	3	0	0	Activation $h_i$
		1	1	0	0	Firing $r_i$

Fig. 2.4

The activation of the four output neurons is 3300, and if we set the threshold of each output neuron to 2, then the output firing is 1100 (where the binary firing rate is 0 if below threshold, and 1 if above). The pattern associator has thus achieved recall of the pattern 1100, which is correct.

We can now illustrate how a number of different associations can be stored in such a pattern associator, and retrieved correctly. Let us associate a new CS pattern 110001 with the UCS 0101 in the same pattern associator. The weights will become as shown next in Fig. 2.5 after learning:

		UCS			
CS		0	1	0	1
		↓	↓	↓	↓
1→		1	2	0	1
1→		0	1	0	1
0→		1	1	0	0
0→		0	0	0	0
0→		1	1	0	0
1→		0	1	0	1

Fig. 2.5

If we now present the second CS, the retrieval is as follows:

		CS				
1→		1	2	0	1	
1→		0	1	0	1	
0→		1	1	0	0	
0→		0	0	0	0	
0→		1	1	0	0	
1→		0	1	0	1	
		↓	↓	↓	↓	
		1	4	0	3	Activation $h_i$
		0	1	0	1	Firing $r_i$

Fig. 2.6

The binary output firings were again produced with the threshold set to 2. Recall is perfect. This illustration shows the value of some threshold non-linearity in the activation function of the neurons. In this case, the activations did reflect some small cross-talk or interference from the previous pattern association of CS1 with UCS1, but this was removed by the threshold operation, to clean up the recall firing. The example also shows that when further associations are learned by a pattern associator trained with the Hebb rule, Eq. 2.2, some synapses will reflect increments above a synaptic strength of 1. It is left as an exercise to the reader to verify that recall is still perfect to CS1, the vector 101010. (The output activation vector  $h$  is 3401, and the output firing vector  $r$  with the same threshold of 2 is 1100, which is perfect recall.)

### 2.3 The vector interpretation

The way in which recall is produced, Eq. 2.3, consists for each output neuron  $i$  of multiplying each input firing rate  $r'_j$  by the corresponding synaptic weight  $w_{ij}$  and summing the products to obtain the activation  $h_i$ . Now we can consider the firing rates  $r'_j$  where  $j$  varies from 1 to  $N$ , the number of axons, to be a vector. (A vector is simply an ordered set of numbers—see

Appendix A1.) Let us call this vector  $r'$ . Similarly, on a neuron  $i$ , the synaptic weights can be treated as a vector,  $w_i$ . (The subscript  $i$  here indicates that this is the weight vector on the  $i$ th neuron.) The operation we have just described to obtain the activation can now be seen to be a simple multiplication operation of two vectors to produce a single output value (called a scalar output). This is the inner product or dot product of two vectors, and can be written

$$h_i = r' \cdot w_i \tag{2.5}$$

The inner product of two vectors indicates how similar they are. If two vectors have corresponding elements the same, then the dot product will be maximal. If the two vectors are similar but not identical, then the dot product will be high. If the two vectors are completely different, the dot products will be 0, and the vectors are described as orthogonal. (The term orthogonal means at right angles, and arises from the geometric interpretation of vectors, which is summarized in Appendix A1.) Thus the dot product provides a direct measure of how similar two vectors are. It can now be seen that a fundamental operation many neurons perform is effectively to compute how similar an input pattern vector  $r'$  is to a stored weight vector. The similarity measure they compute, the dot product, is a very good measure of similarity, and indeed, the standard (Pearson product-moment) correlation coefficient used in statistics is the same as a normalized dot product with the mean subtracted from each vector, as shown in Appendix 1. (The normalization used in the correlation coefficient results in the coefficient varying always between +1 and -1, whereas the actual scalar value of a dot product clearly depends on the length of the vectors from which it is calculated.)

With these concepts, we can now see that during learning a pattern associator adds to its weight vector a vector  $\delta w_i$  that has the same pattern as the input pattern  $r'$ , if the postsynaptic neuron  $i$  is strongly activated. Indeed, we can express Eq. 2.2 in vector form as

$$\delta w_i = k r_i r' \tag{2.6}$$

We can now see that what is recalled by the neuron depends on the similarity of the recall cue vector  $r'$ , to the originally learned vector  $r$ . The fact that during recall the output of each neuron reflects the similarity (as measured by the dot product) of the input pattern  $r'$ , to each of the patterns used originally as  $r'$  inputs (conditioned stimuli in Fig. 2.1) provides a simple way to appreciate many of the interesting and biologically useful properties of pattern associators, as described next.

## 2.4 Properties

### 2.4.1 Generalization

During recall, pattern associators generalize, and produce appropriate outputs if a recall cue vector  $r'$ , is similar to a vector that has been learned already. This occurs because the recall operation involves computing the dot (inner) product of the input pattern vector  $r'$ , with the synaptic weight vector  $w_i$ , so that the firing produced,  $r_i$ , reflects the similarity of the current

input to the previously learned input pattern  $r'$ . (Generalization will occur to input cue or conditioned stimulus patterns  $r'$ , which are incomplete versions of an original conditioned stimulus  $r'$ , although the term completion is usually applied to the autoassociation networks described in Chapter 3.)

This is an extremely important property of pattern associators, for input stimuli during recall will rarely be absolutely identical to what has been learned previously, and automatic generalization to similar stimuli is extremely useful, and has great adaptive value in biological systems.

Generalization can be illustrated with the simple binary pattern associator considered above. (Those who have appreciated the vector description just given may wish to skip this illustration.) Instead of the second CS, pattern vector 110001, we will use the similar recall cue 110100.

		CS				
1→	1	2	0	1		
1→	0	1	0	1		
0→	1	1	0	0		
1→	0	0	0	0		
0→	1	1	0	0		
0→	0	1	0	1		
	↓	↓	↓	↓		
	1	3	0	2	Activation $h_i$	
	0	1	0	1	Firing $r_i$	

Fig. 2.7

It is seen that the output firing rate vector, 0101, is exactly what should be recalled to CS2 (and not to CS1), so correct generalization has occurred. Although this is a small network trained with few examples, the same properties hold for large networks with large numbers of stored patterns, as described more quantitatively in the section on capacity below and in Appendix A3.

### 2.4.2 Graceful degradation or fault tolerance

If the synaptic weight vector  $w_i$  (or the weight matrix, which we can call  $W$ ) has synapses missing (e.g. during development), or loses synapses, then the output activation  $h_i$  or  $h$  is still reasonable, because  $h_i$  is the dot product (correlation) of  $r'$  with  $w_i$ . The result, especially after passing through the output activation function, can frequently be perfect recall. The same property arises if for example one or some of the CS input axons are lost or damaged. This is a very important property of associative memories, and is not a property of conventional computer memories, which produce incorrect data if even only 1 storage location (for 1 bit or binary digit of data) of their memory is damaged or cannot be accessed. This property of graceful degradation is of great adaptive value for biological systems.

We can illustrate this with a simple example. If we damage two of the synapses in Fig. 2.2 to produce the synaptic matrix shown in Fig. 2.8 (where  $x$  indicates a damaged synapse which has no effect, but was previously 1), and now present the second CS, the retrieval is as follows:

		CS				
1→	1	2	0	1		
1→	0	1	0	$x$		
0→	1	1	0	0		
0→	0	0	0	0		
0→	1	$x$	0	0		
1→	0	1	0	1		
	↓	↓	↓	↓		
	1	4	0	2	Activation $h_i$	
	0	1	0	1	Firing $r_i$	

Fig. 2.8

The binary output firings were again produced with the threshold set to 2. The recalled vector, 0101, is perfect. This illustration again shows the value of some threshold non-linearity in the activation function of the neurons. It is left as an exercise to the reader to verify that recall is still perfect to CS1, the vector 101010. (The output activation vector  $h$  is 3301, and the output firing vector  $r$  with the same threshold of 2 is 1100, which is perfect recall.)

### 2.4.3 The importance of distributed representations for pattern associators

A distributed representation is one in which the firing or activity of all the elements in the vector is used to encode a particular stimulus. For example, in the vector CS1 which has the value 101010, we need to know the state of all the elements to know which stimulus is being represented. Another stimulus, CS2, is represented by the vector 110001. We can represent many different events or stimuli with such overlapping sets of elements, and because in general any one element cannot be used to identify the stimulus, but instead the information about which stimulus is present is distributed over the population of elements or neurons, this is called a distributed representation. If, for binary neurons, half the neurons are in one state (e.g. 0), and the other are in the other state (e.g. 1), then the representation is described as **fully distributed**. The CS representations above are thus fully distributed. If only a smaller proportion of the neurons is active to represent a stimulus, as in the vector 100001, then this is a **sparse representation**. For binary representations, we can quantify the sparseness by the proportion of neurons in the active (1) state.

In contrast, a **local representation** is one in which all the information that a particular stimulus or event has occurred is provided by the activity of one of the neurons, or elements in the vector. One stimulus might be represented by the vector 100000, another stimulus by the vector 010000, and a third stimulus by the vector 001000. The activity of neuron or element 1

would indicate that stimulus 1 was present, and of neuron 2, that stimulus 2 was present. The representation is local in that if a particular neuron is active, we know that the stimulus represented by that neuron is present. In neurophysiology, if such cells were present, they might be called 'grandmother cells' (cf. Barlow, 1972, 1995), in that one neuron might represent a stimulus in the environment as complex and specific as one's grandmother. Where the activity of a number of cells must be taken into account in order to represent a stimulus (such as an individual taste), then the representation is sometimes described as using ensemble encoding.

Now, the properties just described for associative memories, generalization and graceful degradation, are only implemented if the representation of the CS or  $r'$  vector is distributed. This occurs because the recall operation involves computing the dot (inner) product of the input pattern vector  $r'$ , with the synaptic weight vector  $w$ . This allows the output activation  $h_i$  to reflect the similarity of the current input pattern to a previously learned input pattern  $r'$  only if several or many elements of the  $r'$  and  $r'$  vectors are in the active state to represent a pattern. If local encoding were used, e.g. 100000, then if the first element of the vector (which might be the firing of axon 1, i.e.  $r'_1$ , or the strength of synapse  $i1$ ,  $w_{i1}$ ) is lost, then the resulting vector is not similar to any other CS vector, and the output activation is 0. In the case of local encoding, the important properties of associative memories, generalization and graceful degradation, do not thus emerge. Graceful degradation and generalization are dependent on distributed representations, for then the dot product can reflect similarity even when some elements of the vectors involved are altered. If we think of the correlation between Y and X in a graph, then this correlation is affected only little if a few X,Y pairs of data are lost (see Appendix A1).

#### 2.4.4 Prototype extraction, extraction of central tendency, and noise reduction

If a set of similar conditioned stimulus vectors  $r'$  are paired with the same unconditioned stimulus  $e_i$ , the weight vector  $w_i$  becomes (or points towards) the sum (or with scaling the average) of the set of similar vectors  $r'$ . This follows from the operation of the Hebb rule in Eq. 2.2. When tested at recall, the output of the memory is then best to the average input pattern vector denoted  $\langle r' \rangle$ . If the average is thought of as a prototype, then even though the prototype vector  $\langle r' \rangle$  itself may never have been seen, the best output of the neuron or network is to the prototype. This produces 'extraction of the prototype' or 'central tendency'. The same phenomenon is a feature of human memory performance (see McClelland and Rumelhart, 1986 Ch. 17, and Section 3.4.4), and this simple process with distributed representations in a neural network accounts for the phenomenon.

If the different exemplars of the vector  $r'$  are thought of as noisy versions of the true input pattern vector  $\langle r' \rangle$  (with incorrect values for some of the elements), then the pattern associator has performed 'noise reduction', in that the output produced by any one of these vectors will represent the true, noiseless, average vector  $\langle r' \rangle$ .

#### 2.4.5 Speed

Recall is very fast in a real neuronal network, because the conditioned stimulus input firing  $r'_j$  ( $j = 1, C$  axons) can be applied simultaneously to the synapses  $w_{ij}$ , and the activation  $h_i$  can

be accumulated in one or two time constants of the dendrite (e.g. 10–20 ms). Whenever the threshold of the cell is exceeded, it fires. Thus, in effectively one step, which takes the brain no more than 10–20 ms, all the output neurons of the pattern associator can be firing with rates that reflect the input firing of every axon. This is very different from a conventional digital computer, in which computing  $h_i$  in Eq. 2.2 would involve  $C$  multiplication and addition operations occurring one after another, or  $2C$  time steps. The brain performs parallel computation in at least two senses in even a pattern associator. One is that for a single neuron, the separate contributions of the firing  $r'_j$  rate of each axon  $j$  multiplied by the synaptic weight  $w_{ij}$  are computed in parallel and added in the same time step. The second is that this can be performed in parallel for all neurons  $i = 1, N$  in the network, where there are  $N$  output neurons in the network. It is these types of parallel processing which enable these classes of neuronal network in the brain to operate so fast, in effectively so few steps.

Learning is also fast ('one-shot') in pattern associators, in that a single pairing of the conditioned stimulus  $r'$  and the unconditioned stimulus  $e$  which produces the unconditioned output firing  $r$  enables the association to be learned. There is no need to repeat the pairing in order to discover over many trials the appropriate mapping. This is extremely important for biological systems, in which a single co-occurrence of two events may lead to learning which could have life-saving consequences. (For example, the pairing of a visual stimulus with a potentially life-threatening aversive event may enable that event to be avoided in future.) Although repeated pairing with small variations of the vectors is used to obtain the useful properties of prototype extraction, extraction of central tendency, and noise reduction, the essential properties of generalization and graceful degradation are obtained with just one pairing. The actual time scales of the learning in the brain are indicated by studies of associative synaptic modification using long-term potentiation paradigms (LTP, see Chapter 1). Co-occurrence or near simultaneity of the CS and UCS is required for periods of as little as 100 ms, with expression of the synaptic modification being present within typically a few seconds.

#### 2.4.6 Local learning rule

The simplest learning rule used in pattern association neural networks, a version of the Hebb rule, is as in Eq. 2.2 above

$$\delta w_{ij} = k r_i r'_j.$$

This is a local learning rule in that the information required to specify the change in synaptic weight is available locally at the synapse, as it is dependent only on the presynaptic firing rate  $r'_j$  available at the synaptic terminal, and the postsynaptic activation or firing  $r_i$  available on the dendrite of the neuron receiving the synapse (see Fig. 2.9b). This makes the learning rule biologically plausible, in that the information about how to change the synaptic weight does not have to be carried from a distant source, where it is computed, to every synapse. Such a non-local learning rule would not be biologically plausible, in that there are no appropriate connections known in most parts of the brain to bring in the synaptic training or teacher signal to every synapse.

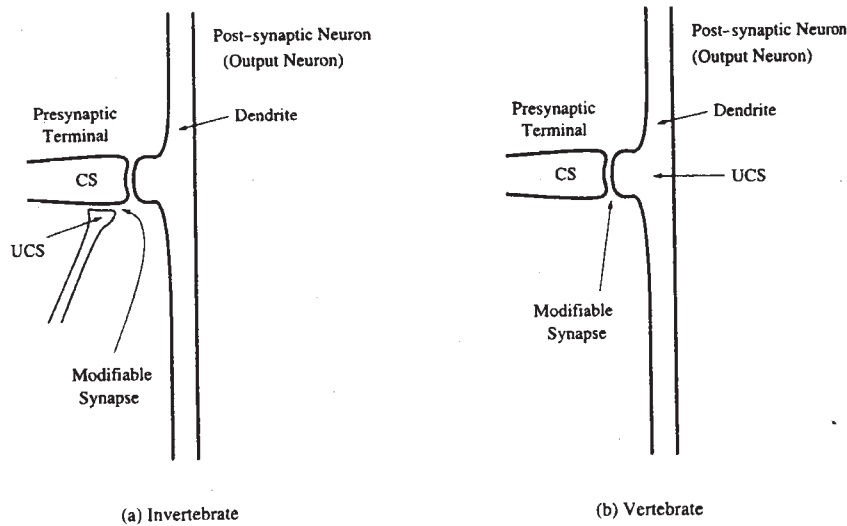


Fig. 2.9 (a) In at least some invertebrate association learning systems, the unconditioned stimulus (UCS) or teaching input makes a synapse onto the presynaptic terminal carrying the conditioned stimulus (CS). (b) In contrast, in vertebrate pattern association learning, the UCS may be made available at all the CS terminals onto the output neuron because the dendrite of the postsynaptic neuron is electrically short, so that the effect of the UCS spreads for long distances along the dendrite.

Evidence that a learning rule with the general form of Eq. 2.2 is implemented in at least some parts of the brain comes from studies of long-term potentiation, described in Chapter 1. Long-term potentiation (LTP) has the synaptic specificity defined by Eq. 2.2, in that only synapses from active afferents, not those from inactive afferents, become strengthened. Synaptic specificity is important for a pattern associator, and most other types of neuronal network, to operate correctly.

Another useful property of real neurons in relation to Eq. 2.2 is that the postsynaptic term  $r_i$  is available on much of the dendrite of a cell, because the electrotonic length of the dendrite is short. Thus if a neuron is strongly activated with a high value for  $r_i$ , then any active synapse onto the cell will be capable of being modified. This enables the cell to learn an association between the pattern of activity on all its axons and its postsynaptic activation, which is stored as an addition to its weight vector  $w_i$ . Then later on, at recall, the output can be produced as a vector dot product operation between the input pattern vector  $r$  and the weight vector  $w_i$ , so that the output of the cell can reflect the correlation between the current input vector and what has previously been learned by the cell.

It is interesting that at least many invertebrate neuronal systems may operate differently from those described here. There is evidence (see e.g. Kandel, 1991) that the site of synaptic plasticity in, for example, *Aplysia* involves the UCS or teaching input having a connection onto the presynaptic terminal which carries the CS (Fig. 2.9a). Conjunctive activity between the UCS and the CS presynaptically alters the effect of the CS in releasing transmitter from the presynaptic terminal. Such a learning scheme is very economical in terms of neurons and synapses, for the conditioning occurs at a single synapse. The problem with such a scheme is that the UCS or teaching input is not made explicit in terms of postsynaptic activation and

is not available at other synapses on the cell. Thus the cell cannot learn an association between a pattern of presynaptic activity  $r'$  and a UCS; and dot product operations, with all their computationally desirable properties such as generalization, completion, and graceful degradation, will not occur in such invertebrate associators. If this arrangement is typical of what is found in associative learning systems in invertebrates, it is a real limitation on their processing, and indeed their computational style would be very different to that of vertebrates. If the associative modification in invertebrates typically involved conjunctions and learning between presynaptic elements, then to arrange a pattern association one might need the rather implausible connectivity shown in Fig. 2.10a, in which the UCS input neuron systematically makes presynaptic teaching connections onto every presynaptic terminal on a particular output neuron, to ensure that the teaching term is expressed globally across the set of inputs to a particular neuron. The implication of this arrangement is that computations and learning in invertebrate learning systems may operate on very different computational lines to those possible in vertebrates, in that the fundamental dot product operation so useful in large neuronal networks with distributed representations may not apply, at least in many invertebrate neuronal systems.

In some other invertebrate neuronal systems, synaptic plasticity may occur between presynaptic and postsynaptic neurons, so a little further comment on differences between at least some invertebrate neuronal systems and the systems found in for example the mammalian cortex may be useful. In some invertebrate systems with relatively small numbers of neurons, the neurons are sufficiently close that each neuron may not only have multiple inputs, but may also have multiple outputs, due to there being many different output

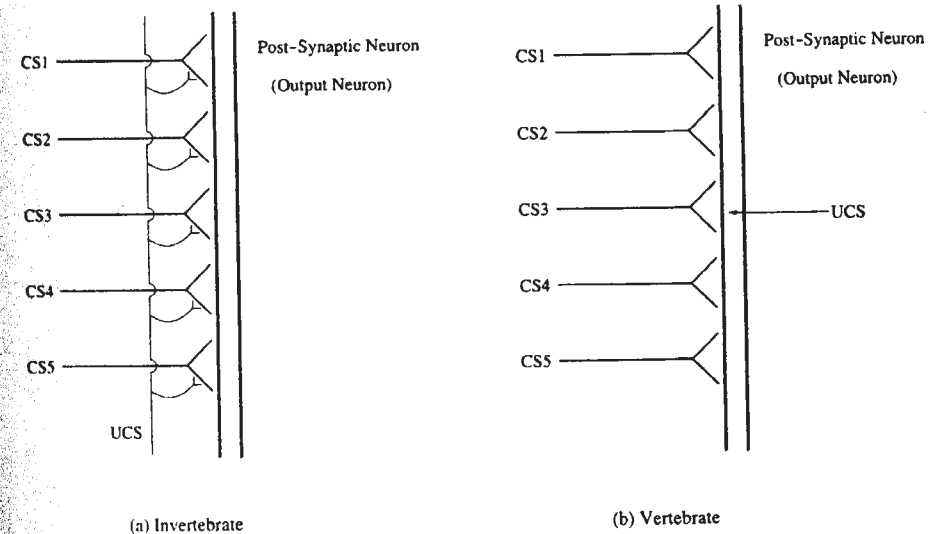


Fig. 2.10 (a) To make a pattern associator from an invertebrate learning system of the type shown in Fig. 2.9a, the UCS neurons would have to make presynaptic teaching terminals onto every CS input terminal on the cell. (b) The situation in vertebrates for a similar function is shown.

connections from different parts of the neuron to different neighbouring neurons. This makes it very difficult to analyse the computations in the networks, as each neuron not only performs local computation on its dendrites (which may occur in vertebrates too), but has many outputs, each difficult to measure. So each invertebrate neuron may operate effectively as a number of different computational units. In contrast, cortical pyramidal cells have one output, which is distributed by (measurable) all-or-none action potentials (spikes) to a large number (typically 10 000–20 000) of other neurons in the network. Thus the computational style of cortical neuronal systems is that there is a large number of neurons with the same type of general connectivity (e.g. neurons in the hippocampal CA3 network), and the computation is performed by a network composed of similar neurons. This enables the computation to be understood and to operate at the whole network level. It enables for example the statistical and analytic approaches to the network properties described in this book to be used. In contrast, in at least the invertebrate systems with relatively small numbers of specialized neurons each with multiple and different outputs, the computations will be much more specific, will not have the same distributed representations and the properties which arise from dot product operation with these, and will be difficult to analyse, especially dynamically. There are in the vertebrate retina (in which the topology makes local computation between neighbouring neurons useful), and olfactory bulb, neurons which do have multiple output connections with graded potentials to neighbouring neurons (see Shepherd, 1990), but these systems are not at all like those in most of the mammalian brain, in for example cortical systems.

### 2.4.7 Capacity

The question of the storage capacity of a pattern associator is considered in detail in Appendix A3. It is pointed out there that, for this type of associative network, the number of memories that it can hold simultaneously in storage has to be analysed together with the retrieval quality of each output representation, and then only for a given quality of the representation provided in the input. This is in contrast with autoassociative nets, in which a critical number of stored memories exists (as a function of various parameters of the network) beyond which attempting to store additional memories results in it becoming impossible to retrieve essentially anything. With a pattern associator, instead, one will always retrieve *something*, but this something will be very little (in information or correlation terms) if too many associations are simultaneously in storage and/or if too little is provided as input.

The conjoint quality–capacity–input analysis can be carried out, for any specific instance of a pattern associator, by using formal mathematical models and established analytic procedures (see e.g. Treves, 1995). This, however, has to be done case by case. It is anyway useful to develop some intuition for how a pattern associator operates, by considering what its capacity would be in certain well-defined simplified cases.

#### Linear associative neuronal networks

These networks are made up of units with a linear activation function, which appears to make them unsuitable to represent real neurons with their positive-only firing rates. However, even

purely linear units have been considered as provisionally relevant models of real neurons, by assuming that the latter operate sometimes in the linear regime of their transfer function. (This implies a high level of spontaneous activity, and may be closer to conditions observed early on in sensory systems rather than in areas more specifically involved in memory.) As usual, the connections are trained by a Hebb (or similar) associative learning rule. The capacity of these networks can be defined as the total number of associations that can be learned independently of each other, given that the linear nature of these systems prevents anything more than a linear transform of the inputs. This implies that if input pattern C can be written as the weighted sum of input patterns A and B, the output to C will be just the same weighted sum of the outputs to A and B. If there are  $N'$  input axons, only at most  $N'$  input patterns are all mutually independent (i.e. none can be written as a weighted sum of the others), and therefore the capacity of linear networks, defined above, is just  $N'$ , or equal to the number of input lines. In general, a random set of less than  $N'$  vectors (the CS input pattern vectors) will tend to be mutually independent but not mutually orthogonal (at  $90^\circ$  to each other). If they are not orthogonal (the normal situation), then the dot product of them is not  $0^\circ$ , and the output pattern activated by one of the input vectors will be partially activated by other input pattern vectors, in accordance with how similar they are (see Eqs 2.5 and 2.6). This amounts to interference, which is therefore the more serious the less orthogonal, on the whole, is the set of input vectors.

Since input patterns are made of elements with positive values, if a simple Hebbian learning rule like the one of Eq. 2.2 is used (in which the input pattern enters directly with no subtraction term), the output resulting from the application of a stored input vector will be the sum of contributions from all other input vectors that have a non-zero dot product with it (see Appendix A1), and interference will be disastrous. The only situation in which this would not occur is when different input patterns activate completely different input lines, but this is clearly an uninteresting circumstance for networks operating with distributed representations. A solution to this issue is to use a modified learning rule of the following form

$$\delta w_{ij} = k r_i (r'_j - x) \quad (2.7)$$

where  $x$  is a constant, approximately equal to the average value of  $r'_j$ . This learning rule includes (in proportion to  $r_i$ ) increasing the synaptic weight if  $(r'_j - x) > 0$  (long-term potentiation), and decreasing the synaptic weight if  $(r'_j - x) < 0$  (heterosynaptic long-term depression). It is useful for  $x$  to be roughly the average activity of an input axon  $r'_j$  across patterns, because then the dot product between the various patterns stored on the weights and the input vector will tend to cancel out with the subtractive term, except for the pattern equal to (or correlated with) the input vector itself. Then up to  $N'$  input vectors can still be learned by the network, with only minor interference (provided of course that they are mutually independent, as they will in general tend to be).

This modified learning rule can also be described in terms of a contingency table (Table 2.1) showing the synaptic strength modifications produced by different types of learning rule, where LTP indicates an increase in synaptic strength (called 'long-term potentiation' in neurophysiology), and LTD indicates a decrease in synaptic strength (called 'long-term depression' in neurophysiology). Heterosynaptic long-term depression is so-called because it is the decrease

Table 2.1 Effects of pre- and postsynaptic activity on synaptic modification

		Postsynaptic activation	
		0	High
Presynaptic firing	0	No change	Heterosynaptic LTD
	High	Homosynaptic LTD	LTP

in synaptic strength that occurs to a synapse which is other than that through which the postsynaptic cell is being activated. This heterosynaptic long-term depression is the type of change of synaptic strength that is required (in addition to LTP) for effective subtraction of the average presynaptic firing rate, in order, as it were, to make the CS vectors appear more orthogonal to the pattern associator. The rule is sometimes called the Singer–Stent rule, after work by Singer (1987) and Stent (1973), and was discovered in the brain by Levy (Levy, 1985; Levy and Desmond, 1985; see Brown *et al.*, 1990). Homosynaptic long-term depression is so-called because it is the decrease in synaptic strength that occurs to a synapse which is (the same as that which is) active. For it to occur, the postsynaptic neuron must simultaneously be inactive, or have only low activity. (This rule is sometimes called the BCM rule after the paper of Bienenstock, Cooper and Munro, 1982, see Chapter 4, on competitive networks.)

### Associative neuronal networks with non-linear neurons

With non-linear neurons, that is with at least a threshold in the activation function so that the output firing  $r_i$  is 0 when the activation  $h_i$  is below the threshold, the capacity can be measured in terms of the number of different clusters of output pattern vectors that the network produces. This is because the non-linearities now present (one per output unit) result in some clustering of all possible (conditioned stimulus) input patterns  $r'$ . Input patterns that are similar to a stored input vector can result due to the non-linearities in output patterns even closer to the stored output; and vice versa sufficiently dissimilar inputs can be assigned to different output clusters thereby increasing their mutual dissimilarity. As with the linear counterpart, in order to remove the correlation that would otherwise occur between the patterns because the elements can take only positive values, it is useful to use a modified Hebb rule

$$\delta w_{ij} = k r_i (r'_j - x).$$

With fully distributed output patterns the number  $p$  of associations that leads to different clusters is of order  $C$ , the number of input lines (axons) per output unit (that is, of order  $N'$  for a fully connected network), as shown in Appendix A3. If sparse patterns are used in the output, or alternatively if the learning rule includes a non-linear postsynaptic factor that is

effectively equivalent to using sparse output patterns, the coefficient of proportionality between  $p$  and  $C$  can be much higher than one, that is many more patterns can be stored than inputs per unit (see Appendix A3). Indeed, the number of different patterns or prototypes  $p$  that can be stored can be derived for example in the case of binary units (Gardner, 1988) to be

$$p \approx C / [a_o \log(1/a_o)] \quad (2.8)$$

where  $a_o$  is the sparseness of the output firing pattern  $r$  produced by the unconditioned stimulus.  $p$  can in this situation be much larger than  $C$  (see Rolls and Treves, 1990, and Appendix A3). This is an important result for encoding in pattern associators, for it means that provided that the activation functions are non-linear (which is the case with real neurons), there is a very great advantage to using sparse encoding, for then many more than  $C$  pattern associations can be stored. Sparse representations may well be present in brain regions involved in associative memory (see Chapters 6 and 7) for this reason.

### 2.4.8 Interference

Interference occurs in linear pattern associators if two vectors are not orthogonal, and is simply dependent on the angle between the originally learned vector and the recall cue or CS vector, for the activation of the output neuron depends simply on the dot product of the recall vector and the synaptic weight vector (Eq. 2.5). Also in non-linear pattern associators (the interesting case for all practical purposes), interference may occur if two CS patterns are not orthogonal, though the effect can be controlled with sparse encoding of the UCS patterns, effectively by setting high thresholds for the firing of output units. In other words the CS vectors need not be strictly orthogonal, but if they are too similar, some interference will still be likely to occur.

The fact that interference is a property of neural network pattern associator memories is of interest, for interference is a major property of human memory. Indeed, the fact that interference is a property of human memory and of network association memories is entirely consistent with the hypothesis that human memory is stored in associative memories of the type described here, or at least that network associative memories of the type described represent a useful exemplar of the class of parallel distributed storage network used in human memory. It may also be suggested that one reason that interference is tolerated in biological memory is that it is associated with the ability to generalize between stimuli, which is an invaluable feature of biological network associative memories, in that it allows the memory to cope with stimuli which will almost never on different occasions be identical, and in that it allows useful analogies which have survival value to be made.

### 2.4.9 Expansion recoding

If patterns are too similar to be stored in associative memories, then one solution which the brain seems to use repeatedly is to expand the encoding to a form in which the different stimuli are less correlated, that is more orthogonal, before they are presented as CS stimuli to a pattern associator. The problem can be highlighted by a non-linearly separable mapping (which captures part of the eXclusive OR (XOR) problem), in which the mapping that is desired is as follows. The neuron has two inputs, A and B (see Fig. 2.11).

Input A	1	0	1
Input B	0	1	1
Required Output	1	1	0

Fig. 2.11

This is a mapping of patterns that is impossible for a one-layer network, because the patterns are not linearly separable (see Appendix A1). There is no set of synaptic weights in a one-layer net that could solve the problem shown in Fig. 2.12. Two classes of patterns are not linearly separable if no hyperplane can be positioned in their  $N$ -dimensional space so as to separate them, see Appendix A1. The XOR problem has the additional constraint that  $A = 0$ ,

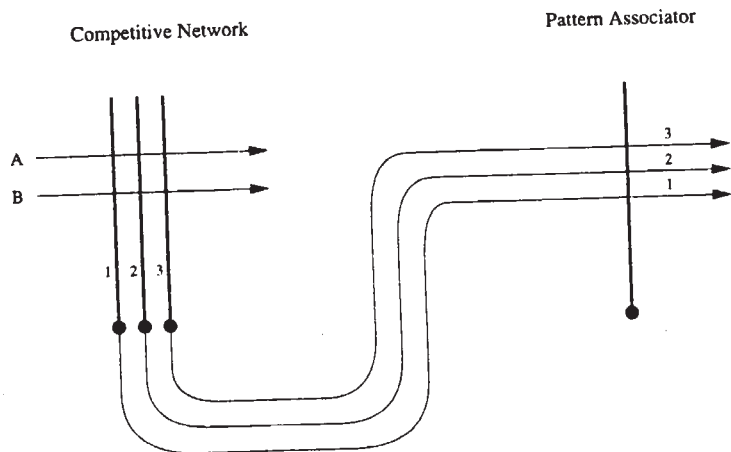


Fig. 2.12 Expansion recoding. A competitive network followed by a pattern associator that can enable patterns that are not linearly separable to be learned correctly.

$B = 0$  must be mapped to Output = 0.) A solution is to remap the two input lines A and B to three input lines 1-3, that is to use expansion recoding, as shown in Fig. 2.12. This can be performed by a competitive network. The synaptic weights on the dendrite of the output neuron could then learn the following values using a simple Hebb rule, Eq. 2.2, and the problem could be solved as in Fig. 2.13.

	Synaptic weight
Input 1 (A = 1, B = 0)	1
Input 2 (A = 0, B = 1)	1
Input 3 (A = 1, B = 1)	0

Fig. 2.13

The whole network would look like that shown in Fig. 2.12.

Expansion encoding which maps vectors with  $N'$  inputs to a set of neurons that is larger than  $N'$  appears to be present in several parts of the brain, and to precede networks which perform pattern association. Marr (1969) suggested that one such expansion recoding was performed in the cerebellum by the mapping from the mossy fibres to the granule cells, which provide the associatively modifiable inputs to the Purkinje cells in the cerebellum (see Chapter 9). The expansion in this case is in the order of 1000 times. Marr (1969) suggested that the expansion recoding was performed by each cerebellar granule cell responding to a low-order combination of mossy fibre activity, implemented by each granule cell receiving 5-7 mossy fibre inputs. Another example is in the hippocampus, in which there is an expansion from the perforant path fibres originating in the entorhinal cortex to the dentate granule cells, which are thought to decorrelate the patterns which are stored by the CA3 cells, which form an autoassociative network (see Chapter 6). The suggestion is that this expansion, performed by the dentate granule cells, helps to separate patterns so that overlapping patterns in the entorhinal cortex are made separate in CA3, to allow separate episodic memories with overlapping information to be stored and recalled separately (see Chapter 6). A similar principle was probably being used as a preprocessor in Rosenblatt's original perceptron (see Chapter 5).

#### 2.4.10 Implications of different types of coding for storage in pattern associators

Throughout this chapter, we have made statements about how the properties of pattern associators, such as the number of patterns that can be stored, and whether generalization and graceful degradation occur, depend on the type of encoding of the patterns to be associated. (The types of encoding considered, local, sparse distributed, and fully distributed, are described in Chapter 1.) We draw together these points in Table 2.2. The amount of information that can be stored in each pattern in a pattern associator is considered in Appendix A3, and some of the relevant information theory itself is described in Appendix A2.

Table 2.2 Coding in associative memories\*

	Local	Sparse distributed	Fully distributed
Generalization, completion, graceful degradation	No	Yes	Yes
Number of patterns that can be stored	$N$ (large)	of order $C/[a_o \log(1/a_o)]$ (can be larger)	of order $C$ (usually smaller than $N$ )
Amount of information in each pattern	Minimal ( $\log(N)$ bits)	Intermediate ( $N a_o \log(1/a_o)$ bits)	Large ( $N$ bits)

\* $N$  refers here to the number of output units, and  $C$  to the average number of inputs to each output unit.  $a_o$  is the sparseness of output patterns, or roughly the proportion of output units activated by a UCS pattern. Note: logs are to the base 2.